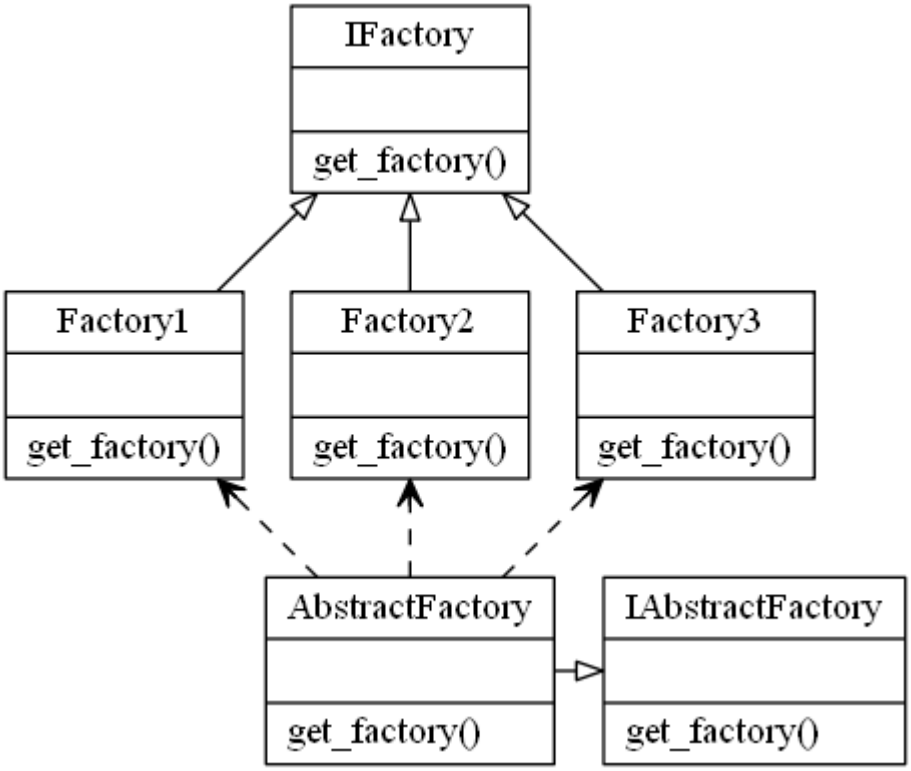
Design Patterns in Python

Abstract Factory Design Pattern

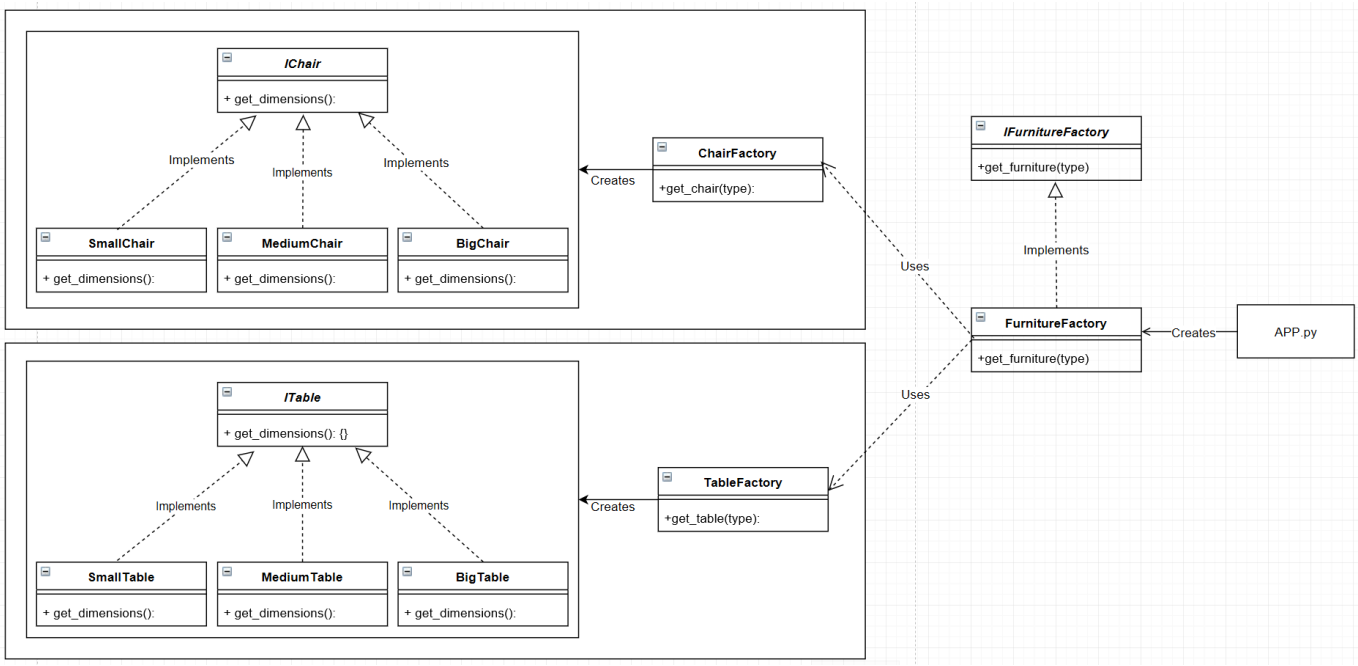
Description

The Abstract Factory Pattern adds an abstract layer over multiple factory method implementations.

The Abstract Factory contains or composites one or more than one factory method



Abstract Factory in the context of a Furniture factory



Source Code

chair_factory.py

```
from abc import ABCMeta, abstractstaticmethod


class IChair(metaclass=ABCMeta):  # pylint: disable=too-few-public-methods
    """The Chair Interface"""

    @abstractstaticmethod
    def dimensions():
        """A static inteface method"""


class BigChair(IChair):  # pylint: disable=too-few-public-methods
    """The Big Chair Concrete Class which implements the IChair interface"""

    def __init__(self):
        self._height = 80
        self._width = 80
        self._depth = 80

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}


class MediumChair(IChair):  # pylint: disable=too-few-public-methods
    """The Medium Chair Concrete Class which implements the IChair interface"""

    def __init__(self):
        self._height = 60
        self._width = 60
        self._depth = 60

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}


class SmallChair(IChair):  # pylint: disable=too-few-public-methods
    """The Small Chair Concrete Class which implements the IChair interface"""

    def __init__(self):
        self._height = 40
        self._width = 40
        self._depth = 40

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}
```

```
class ChairFactory:  # pylint: disable=too-few-public-methods
    """Tha Factory Class"""

    @staticmethod
    def get_chair(chair):
        """A static method to get a table"""
        try:
            if chair == "BigChair":
                return BigChair()
            if chair == "MediumChair":
                return MediumChair()
            if chair == "SmallChair":
                return SmallChair()
            raise AssertionError("Chair Not Found")
        except AssertionError as _e:
            print(_e)
        return None


if __name__ == "__main__":
    CHAIR_FACTORY = ChairFactory().get_chair("SmallChair")
    print(CHAIR_FACTORY.dimensions())
```

table_factory.py

```
from abc import ABCMeta, abstractstaticmethod


class ITable(metaclass=ABCMeta):  # pylint: disable=too-few-public-methods
    """The Table Interface"""

    @abstractstaticmethod
    def dimensions():
        """Get the table dimensions"""


class BigTable(ITable):  # pylint: disable=too-few-public-methods
    """The Big Table Concrete Class which implements the ITable interface"""

    def __init__(self):
        self._height = 60
        self._width = 120
        self._depth = 80

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}


class MediumTable(ITable):  # pylint: disable=too-few-public-methods
    """The Medium Table Concrete Class which implements the ITable interface"""
```

```
    def __init__(self):
        self._height = 60
        self._width = 110
        self._depth = 70

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}


class SmallTable(ITable):  # pylint: disable=too-few-public-methods
    """The Small Table Concrete Class which implements the ITable interface"""

    def __init__(self):
        self._height = 60
        self._width = 100
        self._depth = 60

    def dimensions(self):
        return {"width": self._width, "depth": self._depth, "height":
self._height}


class TableFactory:  # pylint: disable=too-few-public-methods
    """Tha Factory Class"""

    @staticmethod
    def get_table(table):
        """A static method to get a table"""
        try:
            if table == "BigTable":
                return BigTable()
            if table == "MediumTable":
                return MediumTable()
            if table == "SmallTable":
                return SmallTable()
            raise AssertionError("Table Not Found")
        except AssertionError as _e:
            print(_e)
        return None


if __name__ == "__main__":
    TABLE = TableFactory().get_table("SmalTable")
    print(TABLE)
```

furniture_abstract_factory.py

```
from abc import ABCMeta, abstractstaticmethod
from chair_factory import ChairFactory
from table_factory import TableFactory


class IFurnitureFactory(metaclass=ABCMeta):  # pylint: disable=too-few-public-
methods
    """Furniture Factory Interface"""

    @abstractstaticmethod
    def get_furniture(furniture):
        """The static funiture factory inteface method"""


class FurnitureFactory(IFurnitureFactory):  # pylint: disable=too-few-public-
methods
    """The Furniture Factory Concrete Class"""

    @staticmethod
    def get_furniture(furniture):
        """Static get_furniture method"""
        try:
            if furniture in ["SmallChair", "MediumChair", "BigChair"]:
                return ChairFactory().get_chair(furniture)
            if furniture in ["SmallTable", "MediumTable", "BigTable"]:
                return TableFactory().get_table(furniture)
            raise AssertionError("No Furniture Factory Found")
        except AssertionError as _e:
            print(_e)
        return None


FURNITURE = FurnitureFactory.get_furniture("SmallChair")
print(f"{FURNITURE.__class__} : {FURNITURE.dimensions()}")

FURNITURE = FurnitureFactory.get_furniture("MediumTable")
print(f"{FURNITURE.__class__} : {FURNITURE.dimensions()}")
```